John von Neumann Institute for Computing

FEAST: Development of HPC Technologies for FEM Applications

Christian Becker, Sven Buijssen, Hilmar Wobker,
and Stefan Turek

published in

NIC Symposium 2006 ,
G. Münster, D. Wolf, M. Kremer (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **32**, ISBN 3-00-017351-X, pp. 299-306, 2006.

http://www.fz-juelich.de/nic-series/volume32

FEAST: Development of HPC Technologies for FEM Applications

Christian Becker, Sven Buijssen, Hilmar Wobker, and Stefan Turek

Institute for Applied Mathematics and Numerics
University of Dortmund, 44227 Dortmund, Germany
E-mail: feast@math.uni-dortmund.de

One current trend in software development for PDEs, and here especially for FE approaches, clearly goes towards very sophisticated hierarchical techniques and adaptive methods in any sense. In contrast, the employed data and solver structures are mostly chosen as 'globally defined' types which neglect the very specific performance facilities of modern hardware platforms. As a result, the observed computational efficiency is often far from the expected peak rates of (potentially available) several GFLOP/s per processor. These discrepancies, between numerics and software concepts and the available hardware, often lead to unreasonable calculation times for 'real world' problems as can be easily seen from recent benchmark comparisons for commercial as well as research codes. Hence, strategies for massive efficiency enhancement are necessary, not only from the mathematical (algorithms, discretisations) but also from the software side of view. To realise some of these aims our FEM package FEAST ('Finite Element Analysis & Solution Tools') is under development. Recent results on JUMP, including applications from CFD and CSM, are given.

1 Introduction

1.1 Hardware Oriented Numerics

Processor technology is still dramatically advancing and promises further enormous improvements in processing data for the next decade. On the other hand, much lower advances in moving data are expected such that the efficiency of many numerical software tools for PDEs is restricted by the cost of memory access. So, one can state:

- Not data processing, but data moving is costly.
- Employing cache-oriented techniques is a must.
- Exploiting locally structured data is a must.

Examples[1, 2] indicate that many of today's numerical simulation tools – based on the standard sparse MV techniques (see Fig. 1) – are not able to achieve a significant percentage of the high performance on recent processors which is in the range of more than 1 GFLOP/s (Table 1). In the case of fully adaptive FEM codes our measurements show that we better talk about performance rates of 1–10 MFLOP/s for matrix-vector multiplications which are already the fastest components in numerical codes. Complete multigrid solvers often perform even more slowly.

Sparse techniques are basis for most of the recent software packages (Fig. 1). With respect to Table 1 it can be seen that different numberings can lead to identical numerical results and work (w.r.t. arithmetic operations and data accesses) but to huge differences in CPU time. But how to gain more performance? One possibility is to rearrange the data

```
      DO 10 IROW=1,N
      DO 10 ICOL=KLD(IROW),KLD(IROW+1)-1
10    Y(IROW)=DA(ICOL)*X(KCOL(ICOL))+Y(IROW)
```

Figure 1. Standard sparse matrix vector multiplication with indirect adressing in Fortran77 notation

Computer	#unknowns	**CM**	**TL**	**STO**
	33,280	125	105	**100**
Alpha ES40	133,120	81	71	**58**
(667 Mhz)	532,480	60	51	**21**
(1.1 GFLOP/s Peak Linpack)	2,129,920	58	47	**13**
	8,519,680	58	45	**10**

Table 1. Performance rates (MFlop/s) of the FEATFLOW code with different numbering schemes (Cuthill–McKee, TwoLevel, Stochastic) for matrix vector multiplication

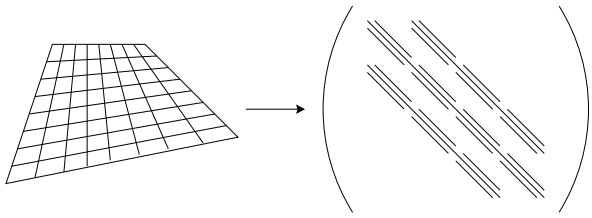

Figure 2. Line- or rowwise numbering of unknowns and resulting matrix structure

structures such that data accesses are more structured and indirect adressing is avoided. Another important topic to consider is cache locality. Modern processor architectures realise most of the possible performance via a sophisticated cache mechanism. To get regular cache-friendly data structures, unknowns should be numerated in a line- or rowwise manner (Fig. 2).

Most finite element discretisations on tensorproduct meshes lead to band structured matrices. The matrix entries are stored in bands of equal size. The matrix vector multiplication is applied bandwise with certain 'windows' to fit cache locality. In case of equidistant meshes band entries are even constant for certain operators, so a complete FEM matrix can be described by a few double values only. Table 2 shows recent results for different architectures (AMD Opteron, NEC vector, JUMP) for some basic operations.

1.2 Solver Schemes: Generalised MG/DD Solvers of SCARC Type

In view of their typically excellent convergence rates, multigrid methods seem to be most suited for the solution of many PDEs. However, as examples[1] have shown, multigrid on general domains has often poor computational efficiency, at least if the implementation is based on standard sparse techniques. Our performance measurements[1] show that realistic MFLOP/s rates for complete multigrid codes are often in the range of 1–10 MFLOP/s only, even on very modern high performance workstations. Moreover, the linear relationship be-

2D case	#unknowns	DAXPY(I)	MV-V	MV-C	MG-TGS-V	MG-TGS-C
Sun V20z	65^2	2172 (633)	1806	3334	1541	2086
(2600 MHz)	257^2	574 (150)	627	2353	751	1423
'**Opteron**'	1025^2	**300 (64)**	**570**	**1774**	**538**	**943**
NEC SX-8	65^2	5070 (1521)	3611	3768	1112	1061
(2000 MHz)	257^2	5283 (1321)	6278	8363	1535	1543
'**Vector**'	1025^2	**5603 (1293)**	**7977**	**15970**	**1918**	**2053**
IBM POWER4	65^2	1521 (845)	2064	3612	906	1071
(1700 MHz)	257^2	943 (244)	896	2896	7111	962
'**JUMP**'	1025^2	**343 (51)**	**456**	**1916**	**438**	**718**

Table 2. Performance rates (MFlop/s) of some basic linear algebra operations: DAXPY: vector vector addition, DAXPY(I) : vector vector addition with indexed factor (sparse matrix multiplication), MV-V/C: matrix vector multication with variable/constant matrix entries, MG-TGS-V/C: complete multi grid cycle with Tri-Gauss-Seidel preconditioner and variable/constant matrix entries

tween problem size and CPU time may sometimes get hardly realisable, due to problem size dependent performance rates of the sparse components. Additionally, the robust treatment of complex mesh structures with locally varying details is often hard to satisfy by typical 'Black Box' components, even for ILU smoothing, and particularly on parallel systems. Motivated by these facts, a more general strategy for solving discretised PDEs is developed (particularly in a parallel framework), which satisfies several conditions:

- The parallel efficiency shall be high due to a non-overlapping decomposition and a low communication overhead.
- The convergence rates ρ are supposed to be independent of the mesh size h, the complexity of the domain and the number of subdomains N, and they shall be in the range of typical multigrid rates (as $\rho \sim 0.1$).
- The method shall be easily implementable and use only existing standard methods.
- The approach shall guarantee treatment of complicated geometries with local anisotropies (huge aspect ratios) without impairment of overall (parallel) convergence rates.

The underlying idea is to 'hide recursively all anisotropies in single subdomains' combined with an outer 'block Jacobi/Gauss-Seidel smoothing' within standard multigrid. This approach is based on the numerical experience[3] that these 'simple' block-oriented schemes perform well as soon as all occurring anisotropies are locally hidden, that means if the local problems on each block are solved (more or less) exactly. These ideas are combined with corresponding hierarchical data and matrix structures, which exploit the described tensorproduct-like meshes on each element of the coarse grid (so-called macro) to achieve the high performance rates for the necessary numerical linear algebra components in the local (multigrid) solvers. Consequently, all solution processes are recursively performed via sequences of more 'local' steps until the lowest level, for instance a single macro with the described generalised tensorproduct mesh, is reached.

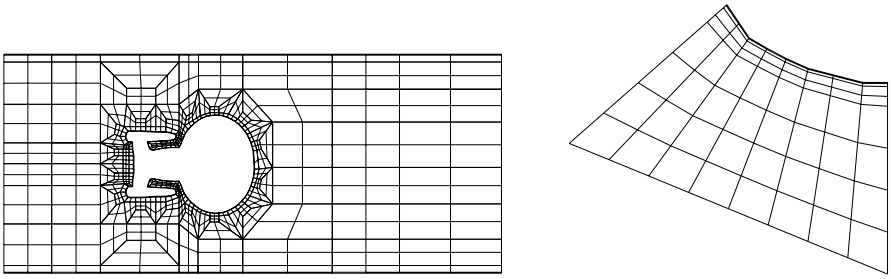

Figure 3. 2D decomposition and zoomed (macro) element (level 3) with locally anisotropic refinement towards the wall

#unknowns	Dirichlet 'Velocity'		Neumann 'Pressure'	
	$AR \approx 10$	$AR \approx 10^6$	$AR \approx 10$	$AR \approx 10^6$
$210,944$	0.17 (**8**)	0.18 (**8**)	0.21 (**9**)	0.15 (**8**)
$843,776$	0.17 (**8**)	0.17 (**8**)	0.20 (**9**)	0.17 (**8**)
$3,375,104$	0.18 (**9**)	0.19 (**9**)	0.22 (**10**)	0.22 (**10**)
$13,500,416$	0.19 (**9**)	0.18 (**9**)	0.23 (**10**)	0.23 (**10**)

Table 3. Global parallel convergence rates and number of SCARC iterations: SCARC-CG solver (smoothing steps: 1 global SCARC; 1 local 'MG-TriGS') for locally (an)isotropic refinement

Consequently, the complete SCARC approach[3] can be characterised as:

- Scalable (with respect to 'quality and number of local solution steps' at each stage)
- Recursive ('independently' for each stage in the hierarchy of partitioning)
- Clustering (for building blocks via 'fixed or adaptive blocking strategies')

Table 3 shows some typical numerical results for a prototypical Poisson problem. The grid for this computation is shown in Fig. 3.

2 Current Research Areas

2.1 FEAST Kernel Development

In this area main work is employed for the optimisation of the basic linear algebra components and generally the optimisation to the JUMP architecture (compiler settings), further the optimisation of the message passing infrastructure for massive parallel computations. Recent results are shown in Table 4.

2.2 Computational Structural Mechanics/ Computational Fluid Dynamics

Aim of this section is to illustrate how problems from CSM and CFD can be tackled in the FEAST framework. Since this basic library only provides facilities to solve scalar problems, the question is how to treat multi-field simulations. The main focus of this

#unknowns	#CPUs	runtime [sec]	(overall) MFLOP/s
3,381,504	8	8.46	927
13,513,216	12	15.28	2031
54,027,264	12	46.61	2281
216,057,856	24	114.24	3709
864,129,024	255	159.42	24276

Table 4. Large scale computations for a prototypical Poisson problem for the NCC configuration (Fig. 3) on JUMP.

section is concentrated on the design of appropriate preconditioners for the resulting saddle point problems which have a major impact on the numerical efficiency of the underlying iterative algorithms.

2.2.1 Generalised Stokes Equation

The incompressible nonstationary Navier–Stokes equations describe the behaviour of a Newtonian fluid at constant temperature with constant kinematic viscosity enclosed in a given volume with Dirichlet and/or Neumann boundary conditions. Neglecting the nonlinear convection term and applying a simple time-discretisation method with timestep k leads to the generalised Stokes equation:

$$\mathbf{u} - \nu k \Delta \mathbf{u} + \nabla p = \mathbf{f}, \qquad \nabla \cdot \mathbf{u} = 0 \tag{1}$$

A similar equation arises in CSM: One possibility to address the problem of *nearly incompressible* elastic material is to introduce, beside the displacements $\mathbf{u}$, a second variable $p := -\lambda \nabla \cdot \mathbf{u}$, which results in a mixed formulation. When a Newmark time discretisation scheme is applied, it comes to the following generalised equation

$$\mathbf{u} - 2\mu\, \tilde{k}\, \nabla \cdot \varepsilon(\mathbf{u}) + \nabla p = \mathbf{f}, \qquad \nabla \cdot \mathbf{u} + \frac{1}{\lambda} p = 0, \tag{2}$$

with $\tilde{k} := \beta k^2$ and β coming from the Newmark scheme. Due to the similarity between equation (1) and (2) we will concentrate only on the Stokes equation from now on. Most of the following applies to the elasticity case, as well, while differences will be emphasised.

At present, FEAST and its underlying SPARSEBANDEDBLAS library only feature discretisation with bilinear elements. Since a straight-forward discretisation with bilinear elements for both velocity and pressure (Q_1/Q_1) would violate the so-called Babuška-Brezzi condition appropriate stabilisation is needed.[4,5] In order not to lose the ability of dealing with irregular grids we extend the standard stabilisation technique by considering directional derivatives

$$c(p,\psi) = \sum_K (h_K^\xi)^2 \left(\xi \nabla p, \xi \nabla \psi\right)_K + (h_K^\eta)^2 \left(\eta \nabla p, \eta \nabla \psi\right)_K$$

where h_K^ξ, h_K^η measure the extensions of each element K for the local coordinate system (ξ, η). After discretisation the problem is brought down to repeatedly solving linear systems of the following type:

$$\begin{pmatrix} A & B \\ B^{\mathsf{T}} & C \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{3}$$

2.2.2 Solving Strategies for Saddle Point Problems

Disregarding the matrix C consisting of stabilisation terms (and the compressibility constraint in the elasticity case) equation (3) is a classic saddle point problem. For nonsingular matrices A the velocity u can be eliminated formally, yielding the scalar equation

$$\left(B^{\mathsf{T}} A^{-1} B - C\right) p = B^{\mathsf{T}} A^{-1} f - g \tag{4}$$

with the so called *pressure Schur complement* $S := B^{\mathsf{T}} A^{-1} B - C$. This equation is solved with a preconditioned Krylov-space method.

Within such a method matrix-vector-multiplications with S have to be performed. As S is only given implicitly this means three matrix-vector-multiplications and "inverting" the matrix A. The latter has to be done exactly — at least in the first iterations — to maintain S-orthogonality of the iterates. The inversion of A can be overcome if the algorithm is embedded in an outer defect correction method acting on the whole system (3). The corresponding basic iteration looks like:

$$\begin{pmatrix} u^{n+1} \\ p^{n+1} \end{pmatrix} = \begin{pmatrix} u^n \\ p^n \end{pmatrix} + \mathcal{N}_S^{-1} \left[\begin{pmatrix} f \\ g \end{pmatrix} - \begin{pmatrix} A & B \\ B^{\mathsf{T}} & C \end{pmatrix} \begin{pmatrix} u^n \\ p^n \end{pmatrix} \right] \tag{5}$$

Thus the Schur complement method merely acts as a preconditioner (formally written as $\mathcal{N}_S^{-1}$), which allows the approximate treatment of A^{-1}. The basic iteration (5) is realised as Krylov-space method.

A second approach is to choose in the basic iteration (5) the block triangular matrix

$$\mathcal{N} := \begin{pmatrix} A & 0 \\ B^{\mathsf{T}} & -S \end{pmatrix}. \tag{6}$$

as *block-preconditioner* for the whole system (3). For the preconditioned system matrix it can easily be shown that the corresponding Krylov subspace has dimension 2, i. e. the solution of the preconditioned system would require only two iterations of a Krylov-space method.

The application of $\mathcal{N}^{-1}$, which involves the exact computation of A^{-1} and S^{-1}, is much too expensive, such that (6) is actually replaced by

$$\tilde{\mathcal{N}} := \begin{pmatrix} \tilde{A} & 0 \\ B^{\mathsf{T}} & -\tilde{S} \end{pmatrix}, \tag{7}$$

where $\tilde{A}$ and $\tilde{S}$ denote preconditioners for A and S, respectively. While the design of $\tilde{S}$ requires a closer look at the underlying equations, which will be done in the next section, the realisation of $\tilde{A}$ is straightforward: In the Stokes case, A consists of two non-zero block matrices L_1, L_2 on the diagonal (discretisations of scalar Laplace operators for the x- and the y-component) and zero off-diagonal block matrices. So, the preconditioner $\tilde{A}$ is simply realised, e. g. by independently doing one SCARC iteration for each component. In the elasticity case, however, x- and y-direction are coupled, resulting in non-zero off-diagonal blocks in A. Consequently, we cannot simply do two independent SCARC iterations as in Stokes case, but we have to resolve the coupling by embedding the SCARC solves as preconditioner into another outer Krylow-space method applied to A.

Anyway, in both cases the treatment of a multi-dimensional system is brought down to the solution of scalar equations, which enables us to exploit the SCARC solvers' strengths.

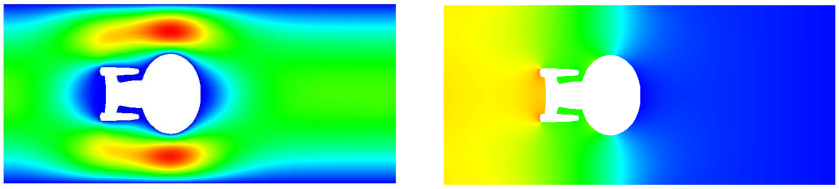

Figure 4. Example calculation for Stokes problem, velocity and pressure, NCC grid (see Fig. 3)

#CPUs	#unknowns	#vertices	runtime [sec]	(overall) MFLOP/s
51	2,540,928	846,976	560.47	291
103	2,540,928	846,976	556.07	293
103	10,144,512	3,381,504	1280.34	804
206	162,081,792	54,027,264	1095.13	7639

Table 5. Results for Stokes simulation on JUMP

2.2.3 Preconditioning of the Schur Complement

In both approaches to solve the system (3) we face the problem that a preconditioner $\tilde{S}$ for the Schur complement $S = B^{\mathsf{T}}A^{-1}B - C$ is needed. Examining the generalised Stokes equation (1) we can deduce the structure of the system matrix $A = M + \nu kL$, where M is the (lumped) mass matrix and L the Laplacian, both block-structured with zero off-diagonals. The "nature" of A clearly depends on the size of the timestep k: For very small timesteps the mass matrix dominates, while it has, in fact, no influence for very large timesteps and even vanishes for the stationary Stokes case. To construct a preconditioner that efficiently covers the whole range of relevant timesteps we exploit the additive decomposition of A. We design the preconditioner for the distinct parts of the Schur complement S. The *reactive part*, $B^{\mathsf{T}}M_l^{-1}B$, can be interpreted as a discretisation matrix steeming from a mixed formulation of the (continuous) Poisson problem. So, the preconditioning operator is chosen as L_p, the Laplacian matrix corresponding to the discrete pressure space. The continous operator associated with the *diffusive part*, $B^{\mathsf{T}}L^{-1}B$, is spectrally equivalent to the identity[6], so $M_{l,p}$, the lumped pressure mass matrix, is an optimal preconditioner. This also holds for the elasticity case, where we have, instead of νkL, the matrix $2\mu\tilde{k}K$ with K being the discretisation of $\nabla \cdot \varepsilon(\mathbf{u})$. A linear combination of the two parts gives the desired Schur complement preconditioner:

$$\tilde{S}^{-1} = L_p^{-1} + \nu k M_{l,p}^{-1} \tag{8}$$

This preconditioner seems not to cover the matrix C from (4). Its entries are of magnitude $O(h^2)$. Only if the time step k is about the size of h^2 or smaller it has to be incorporated. In the elasticity case, the part of C coming from the compressibility constraint is simply a pressure mass matrix and thus can be covered by the diffusive part of the preconditioner (8).

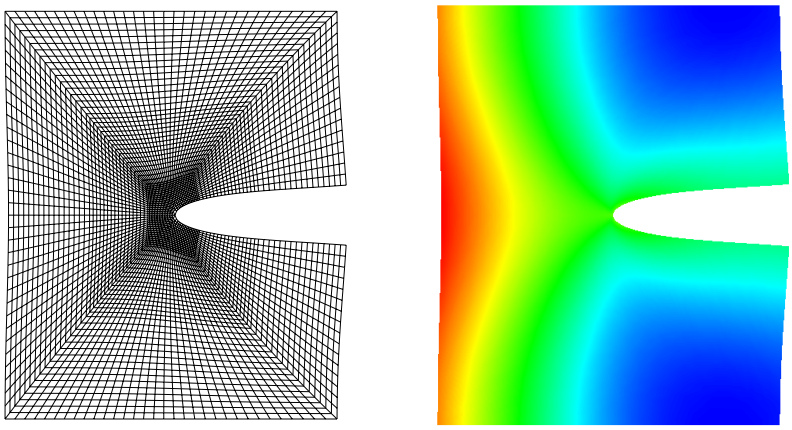

Figure 5. Computational domain and displacement (x direction) field for example elasticity problem

#elements	runtime [sec]	(overall) MFLOP/s
327,680	3.66	1382
1,310,720	9.77	2046
5,242,880	29.60	2683
20,971,520	129.13	2455

Table 6. Results for example elasticity problem on JUMP with 20 CPUs

References

1. S. Turek, A. Runge, and Ch. Becker. The feast indices – realistic evaluation of modern software components and processor technologies. *Computers and Mathematics with Applications*, 41:1431–1464, 2001.
2. Ch. Becker, S. Kilian, and S. Turek. Hardware–oriented numerics and concepts for PDE software. In *FUTURE 1095*, pages 1–23. Elsevier, 2003. International Conference on Computational Science ICCS2002, Amsterdam.
3. S. Kilian. *ScaRC als verallgemeinerter Mehrgitter– und Gebietszerlegungsansatz für parallele Rechnerplattformen*. Logos Verlag, Berlin, 2002. ISBN 3-8325-0092-8.
4. L. P. Franca and Th. J. R. Hughes. A new finite element formulation for computational fluid dynamics: VII. The Stokes–problem with various well–posed boundary conditions: Symmetric formulations that converge for all velocity/pressure spaces. *Computer Methods in Applied Mechanics and Engineering*, 65:85–96, 1987.
5. R. Becker. *An Adaptive Finite Element Method for the Incompressible Navier—Stokes Equations on Time–Dependent Domains*. PhD thesis, Universität Heidelberg, 1995.
6. S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999.